

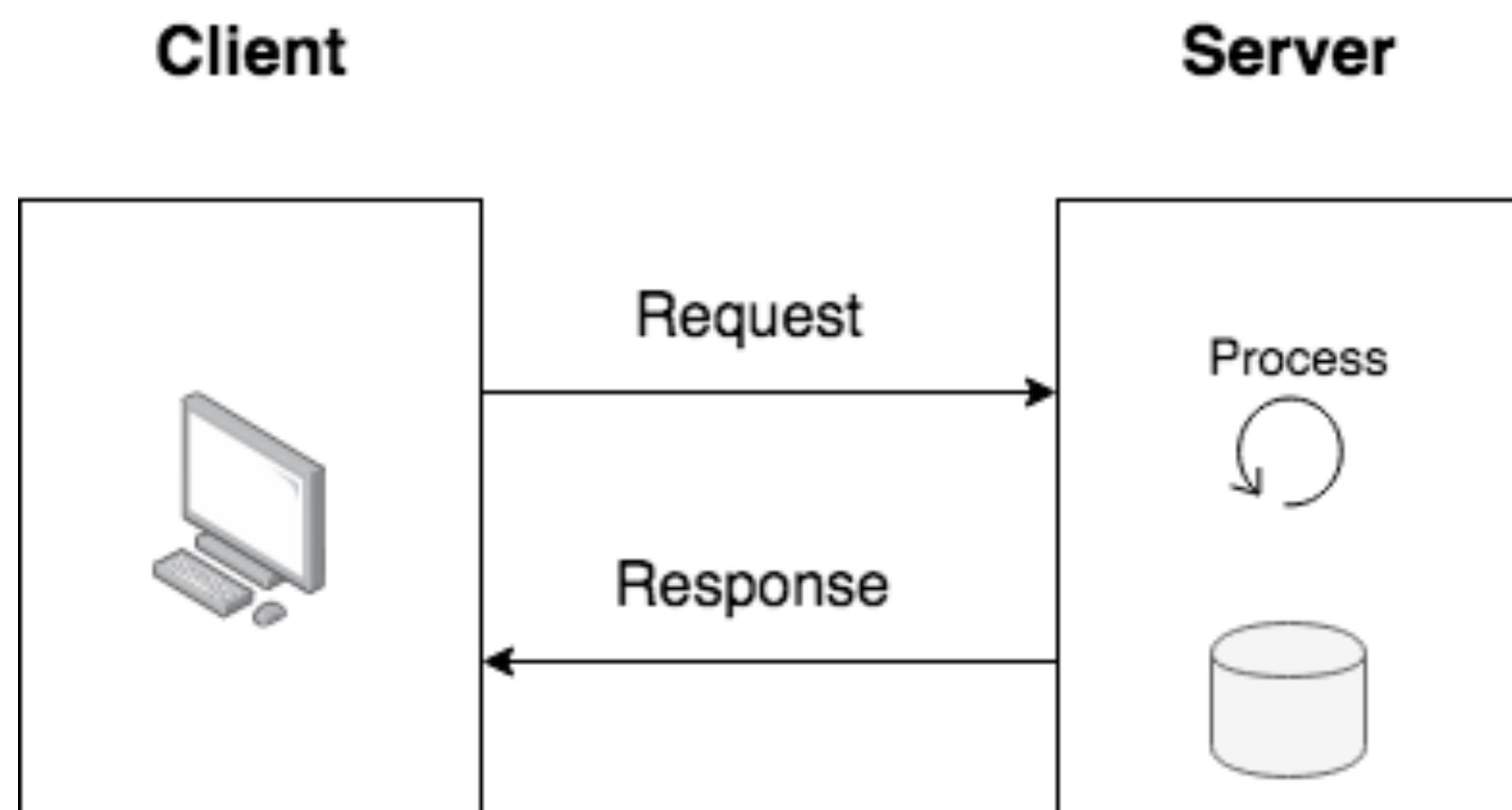
# Aplicações em tempo real

Utilizando socket.io

# WebSockets

- Protocolo utilizado para construção de aplicações real-time como chat, transmissões de conteúdos...;
- Conexão persistente com o backend, estabelecendo um canal bidirecional de comunicação;
- Possibilita que o servidor envie respostas sem necessariamente uma chamada client-side.

# Protocolo HTTP vs WebSocket



Client

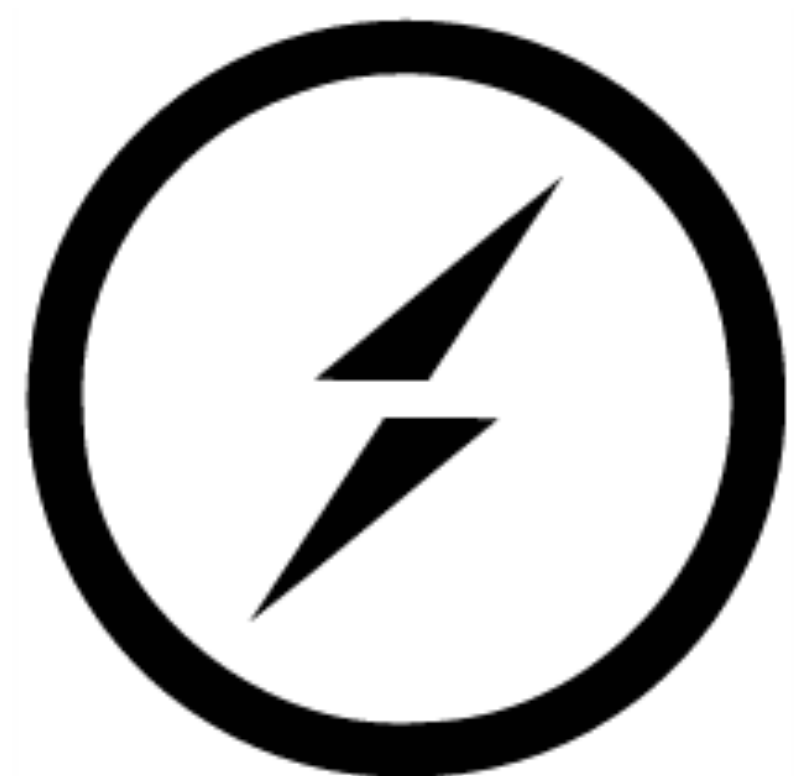


Server

# Casos de uso

- Sistemas em tempo real - transmissão de jogos, sistema de notificações;
- Edição colaborativa - google docs e qualquer outro sistema similar;
- Jogos multiplayer;
- Aplicações de Chat - messenger, whatsapp, telegram...

# Implementação com Node.js



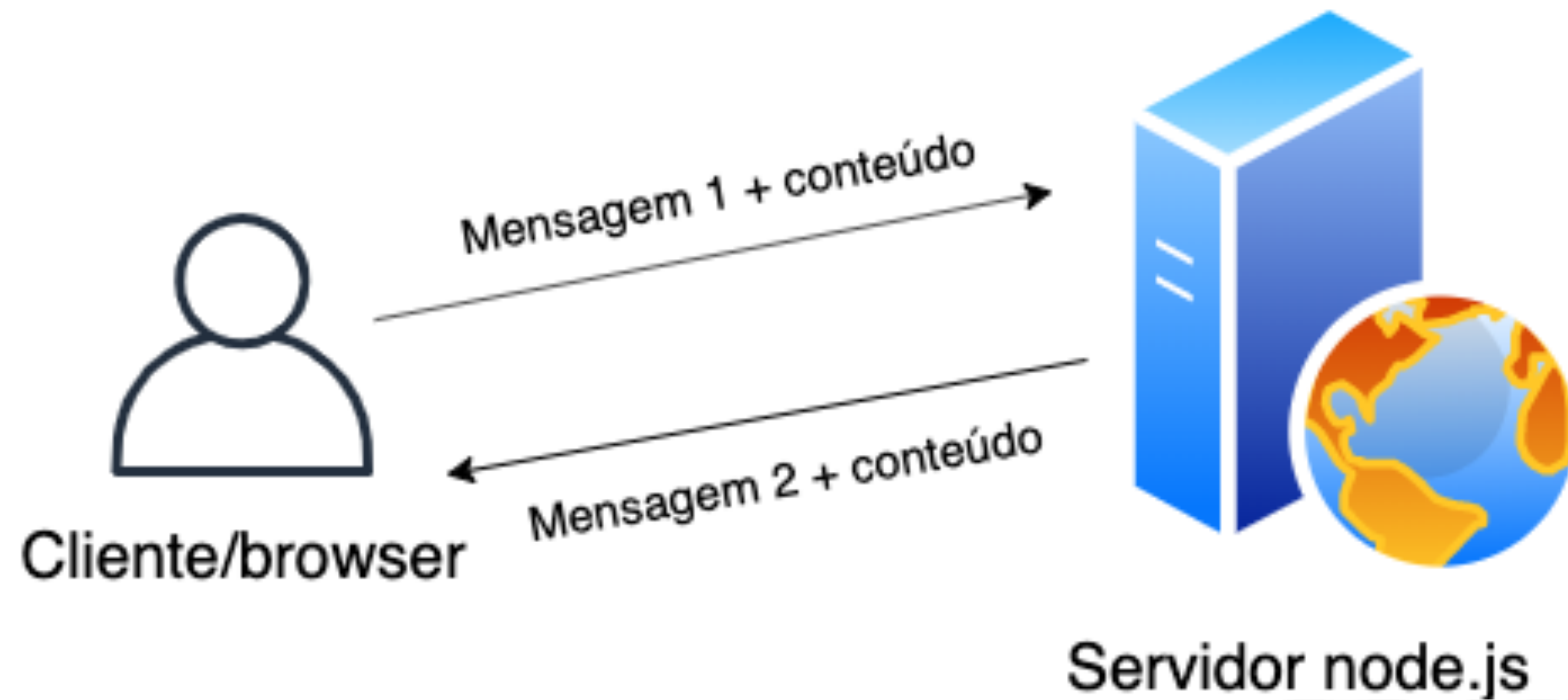
**socket.io**



# Socket.io


- Interface com um conjunto de métodos para criação de websockets em aplicações Node;
- Funciona tanto no cliente quanto no servidor;
- Arquitetura orientada a eventos - cada mensagem recebida e cada usuário conectado na aplicação emite um evento com as informações necessárias para tratamento do cenário em questão.

# Socket.io



# Socket.io - evento de conexão

- Inicia um listener no evento “connection” para tratar os callbacks de quando um usuário se conecta na instância do servidor;



```
1 io.on('connection', socket => {  
2   console.log(`Usuário conectado com o id: ${socket.id}`)  
3 })
```



# Socket.io - evento de conexão (client-side)

- Instancia uma conexão websocket na URL especificada



```
1 <script src="/socket.io/socket.io.js"></script>
2 <script>
3   const socket = io('http://localhost:3000');
4 </script>
```

# Socket.io - senders

- Os senders possuem a mesma sintaxe tanto no cliente quanto no servidor;
- Emitem eventos bidirecionais que podem passar qualquer tipo de conteúdo: strings, números, objetos, arrays, arrays de objetos e assim por diante.



```
1 // Strings
2 socket.emit('mensagem', 'Conteúdo da mensagem')
3
4 // Objetos
5 socket.emit('mensagem', { author: 'Samuel', bio: 'Hello world!' })
6
7 // Arrays
8 socket.emit('mensagem', [{ text: 'Hello' }, { text: 'World' }])
```

# Socket.io - receivers

- Os receivers também possuem a mesma sintaxe tanto no cliente quanto no servidor;
- Inicializam listeners nos eventos conhecidos (senders - que podem ser enviados tanto do lado do cliente quanto do lado do servidor);
- Utilizam funções de callbacks que possibilitam tratar os eventos recebidos

# Socket.io - receivers



```
1 socket.on( 'mensagem', ( conteudo ) => {  
2   console.log( conteudo )  
3 } );
```



# Socket.io - broadcasting

- A função ***broadcast.emit*** emite evento para todas as instâncias conectadas no servidor, exemplo:
  - Três usuários se conectam no servidor, em um chat;
  - Um deles envia uma mensagem para a sala;
  - O servidor pode “ecoar” a mensagem para todas as conexões ativas no chat;



```
1 io.on('connection', (socket) => {  
2     socket.on('mensagem', conteudo => {  
3         socket.broadcast.emit('mensagemParaTodos', conteudo)  
4     })  
5 })
```

# Exercício 5